High-Level Access APIs in the OSIMIS TMN Platform: Harnessing and Hiding

George Pavlou, Thurain Tin - University College London, UK
Andy Carr - Cray Communications, UK

Abstract. There is a common unjustified belief that OSI management technology, despite being very powerful, is difficult to implement because of the complexity of the underlying service/protocol (CMIS/P) and the power and expressiveness of the associated information model. Industry initiatives to standardise Application Program Interfaces (APIs) in this area have drawn the line too low, precisely at the management service level, resulting in the daunting to use XOM/XMP API. On the contrary, the OSIMIS TMN platform proposes much higher level APIs, both for realising new objects in agent applications and for accessing those objects in a distributed fashion. The latter are the Remote and Shadow MIB APIs and the concepts and rationale behind them are laid out while it is explained how object-oriented technology can be used to harness and hide, retaining at the same time the full power of the underlying service.

1. Introduction

The OSI Management Model [X701] forms the basis for the Telecommunication Management Network (TMN) [M3010], both for intra- (Qx/Q3 interfaces) and inter-TMN (X interface) communication. Following a fully fledged object-oriented approach, it lends itself naturally to object-oriented realisation but it seems that industry is reluctant for the jump from modular/object-based to fully object-oriented technology. Because of this nature and its undoubtfully rich functionality, it is considered difficult to implement and has not yet enjoyed widespread support. The important aspect to accelerate its acceptance and use is the provision of infrastructure that packages parts of its functionality behind well-defined Application Program Interfaces (APIs). This will enable the existence of infrastructure from independent vendors, reducing development costs and making possible the easy migration to another vendor's package for performance, cost or other reasons, avoiding the current monolithic realisation approach.

The key question to be asked in the case of OSI management is where exactly the line between generic and specific infrastructure should be drawn in order to result in industry-standard APIs. Initiatives in this area have drawn the line too low, exactly at the management service level [X710]. The current widely acceptable API is X/Open's XOM/XMP [XOpen], XOM being the abstract syntax API while XMP the management service one, commonly designed for both the OSI CMIS and the Internet SNMP. Though this allows for the provision of conformant management protocol stacks, it does very little to help implementors of management applications with the object-oriented aspects of both managed object realisation (agent) and high-level distributed access (manager) aspects. In addition, it is rather daunting to use, its complexity owing much to the commonality with SNMP, an ill-conceived idea due to their different nature, and the bulky and inefficient vendor independent XOM API.

On the other hand, the OSIMIS TMN platform [Pav93a][Pav93b] was conceived as high-level object-oriented infrastructure that facilitates the development of management applications based on the OSI model, hiding protocol related aspects and bridging the gap between the open and distributed systems worlds. It provides extensive support for agent, manager and hybrid applications, supporting manager access APIs at three levels: The procedural CMIS API, known as *MSAP* (Management Service Access Point), similar to XOM/XMP in functionality but much simpler. The *Remote MIB* (*RMIB*) API which provides the object-oriented abstraction of a management association in terms of a local agent in the accessing application for a physically remote MIB. Finally, the *Shadow MIB (SMIB)* API which provides the abstraction of shadow copies of remote managed objects, offering "standardised" local storage and enabling automatic stateful cacheing strategies to be applied.

The rationale and concepts behind the last two are explained here, showing how object-oriented technology can be used to harness and hide, retaining at the same time the full power of the underlying CMIS service. An overview of the OSIMIS platform is given first, putting those two APIs into perspective with respect to the rest of the infrastructure. The "raw" CMIS MSAP API is then discussed in conjunction to high-level abstract syntax support which is fundamental to higher level APIs. The issues behind the RMIB and SMIB access APIs are then explained and the approach as a whole is finally discussed.

2. The OSIMIS Platform Architecture

The OSIMIS TMN platform is object-oriented software infrastructure with C++ [Strou] APIs that enables the quick and efficient development of management applications of any type: Network Element (NE) agents, Q-Adaptors (QAs), Mediation Devices (MDs), Operations Systems (OSs) and Workstations (WSs). It is based on the OSI model whose power and complexity hides behind Object-Oriented APIs, enabling application implementers to concentrate in management policies rather than be burdened with the details of management information access. A generic gateway (QA) facility between CMIS and SNMP is also provided for automatic SNMP M to Qx interface conversion, addressing the ubiquity of SNMP-capable NEs.

The OSIMIS layered architecture and generic applications are shown in Figure 1. The OSI stack up to the ACSE/ROSE level is provided by ISODE [ISODE] OSI stack, while the Internet stack up to UDP is provided by the UNIX operating system kernel. On top of these OSIMIS provides procedural (i.e. non-OO) realisations of CMIS/P and SNMP, the CMIS API being the aforementioned MSAP, complemented by the ISODE Presentation Element (PE) ASN.1 API. Work is ongoing to support the industry standard XOM/XMP API, which will make OSIMIS stack independent. This change will be invisible to applications as CMIS is hidden by the higher level Generic Managed System (GMS) agent and RMIB/SMIB manager APIs while ASN.1 is hidden by the high-level abstract syntax API which currently encapsulates the ISODE PE and in the future the XOM one. There is similar infrastructure to RMIB to facilitate SNMP object access *(RMIB')*. Finally, the Directory Support Service (DSS) provides access to the X.500 Directory [X.500] for address resolution and location and other transparency services. While the above infrastructure is realised as libraries linked with applications, OSIMIS provides also a number of fundamental generic applications. These are ASN.1 and GDMO compilers with C++ bindings, a set of generic managers including a dynamic MIB browser, the CMIS/P-SNMP gateway supported by a information model translator and a generic Directory Service Agent (DSA) with a set of Directory User Agents (DUAs), the last two provided by ISODE.

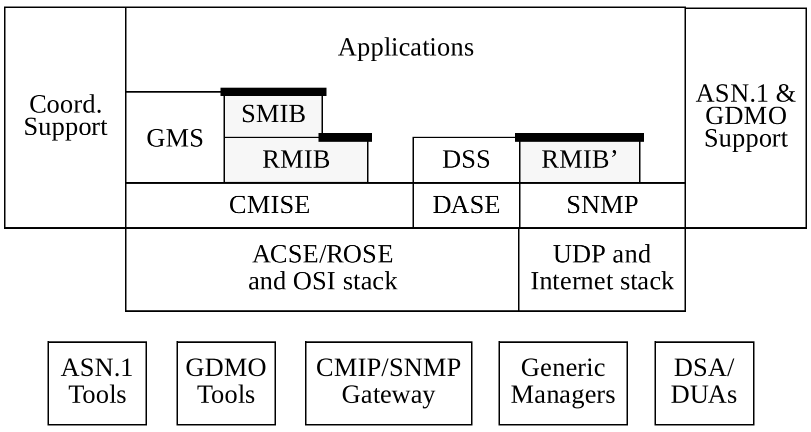

Fig. 1. The OSIMIS Layered Architecture and Generic Applications

A fundamental element for complex applications e.g. an OS that acts in both manager and agent roles is the coordination with respect to internal and external events. OSIMIS provides an object-oriented coordination mechanism that supports real-time "wake-ups" and a first-come-first-served event-driven mechanism. This works best for asynchronous external communications in order to avoid blocking and as such all the higher level APIs have both synchronous and asynchronous options, as is the case with the RMIB and SMIB ones. This does not preclude the use of co-routine or thread mechanisms in which case only synchronous interfaces may be used as they are easier to program - no maintenance of state required by applications. Co-routines have already been used and threads are planned for the future.

3. Management Service and High-Level Abstract Syntax Support

The OSIMIS CMIS MSAP API was conceived much before standard APIs such as the X/Open XOM/XMP were specified and as such it does not conform to the latter. Having been designed specifically for CMIS and not for both CMIS and SNMP, it hides more information and may result in more efficient implementations. The reason it is a procedural object-based in the C programming language and not a fully object-oriented implementation in C++ like the rest of OSIMIS, is in order to conform to the ISODE style, the trend in industry APIs and to be easily integrated by diverse environments. It offers full control over every CMIS detail, leaving its user to deal with associations, assemble linked replies, handle errors and encode/decode attribute, action and event report values. CMIS requests and responses are modelled through procedure calls while incoming indications and confirmation are modelled through a single *"wait"* call, dealing with all possible situations. Multiplexing of incoming data on various associations is an orthogonal issue and can be handled either by the OSIMIS coordination support or any other user-defined mechanism.

Programming applications at this level can be tedious and error-prone and clearly higher level abstractions are necessary. One issue though which is common to all the higher-level APIs is abstract syntax support. Ideally, a distributed management application programmer should be completely shielded from encoding/decoding details and should be allowed to program in terms of internal representations e.g. C or C++ data types, being unaware of the underlying abstract/transfer syntax mechanism. This is possible through object-oriented ASN.1 compilers like the one provided by OSIMIS, which produces C++ objects of a certain style while it can be modified to work with a different underlying mechanism. Currently the ISODE PE ASN.1 API is supported while the X/Open XOM one will be supported in the future. This is achieved by using polymorphism to encapsulate behaviour in data types with respect to encoding and decoding, which take place in a totally transparent fashion as requested by the infrastructure. This OO ASN.1 API is used by all the high-level OSIMIS APIs, namely the GMS, RMIB and SMIB, giving a distributed system programming flavour and dispensing with explicit ASN.1 manipulation.

4. The Remote MIB Access API

The object-oriented and expressive nature of the OSI Management Model [X701] and the associated information model [X720] provide the essential framework through which management applications exchange management information using the CMIS service. Though the OSI structure of management information is largely based on object-oriented design and specification methodology, the same is not true for the level of programming support provided by most CMIS APIs as it is the case with both the OSIMIS MSAP and the X/Open XOM/XMP ones.

Most CMIS/P API realisations leave the implementor to deal with the unpleasant, low-level mechanics of management information access. For example, object identifiers are used for class, attribute, action and event report names while distinguished names in their native forms are used to address object instances. Similarly, scoping and filtering mechanisms that are used to select and eliminate the desired objects for the operation are tied down to low-level data structures. Furthermore, the CMIP protocol is designed in such a way that only a single instance information can be carried in the response Protocol Data Unit (PDU). Hence, in the case where multiple managed object replies are returned as a result of a single request, the response PDUs are linked with a common identifier and the task of assembling those PDUs into managed object results falls, unfortunately, to the hands of the application implementor.

In the RACE NEMESYS and ICM projects, studies on the design and construction of high-level access APIs have been conducted to provide efficient access to the remote management information bases, supporting at a higher level the development of applications in manager roles. The Remote MIB (RMIB) Access API has culminated from one such study, offering a high-level friendly CMIS interface. The primary motive behind the RMIB API has been the desire to “hide” as much as possible of the low-level details of CMIS/P as mentioned, using object-oriented abstractions whilst retaining its full power. The advantages gained from the implementor’s point of view are two-fold: firstly s/he will be freed from the direct manipulation of unfriendly and terse management parameters, and hence be able to concentrate on realising the management policy i.e. the application’s intelligence. Secondly, the reduction in application code size means that development can be more rapidly accomplished.

The RMIB Access API provides a high-level object-oriented abstraction of remote OSI MIBs using the notion of an “association object”. This object is used to encapsulate the management association with a remote agent, hiding the use of the underlying low-level CMIS/P parameters and access to the remote Management Information Tree (MIT). The normal procedures of association control are hidden through friendlier means using only the logical application name and host information and exploiting possibly the location transparency support service. In realisation terms, the association object is represented in a C++ class called *RMIBAgent*. Each instantiation of this class will allow an association establishment with one remote agent. In addition to association control, the RMIBAgent class defines a set of high-level CMIS-like messages at the API, concealing the actual CMIS/P calls. Management operations are thus performed by sending the appropriate messages to the association object. Figure 2 shows the model of the general interaction involved. Hence, this approach allows an association object to be flexibly manipulated among other objects in the application. The level of support provided by the RMIBAgent class, therefore, enables the implementor to think and design the application in terms of abstractions, taking into account the clear separation of concern from the raw low-level access details.

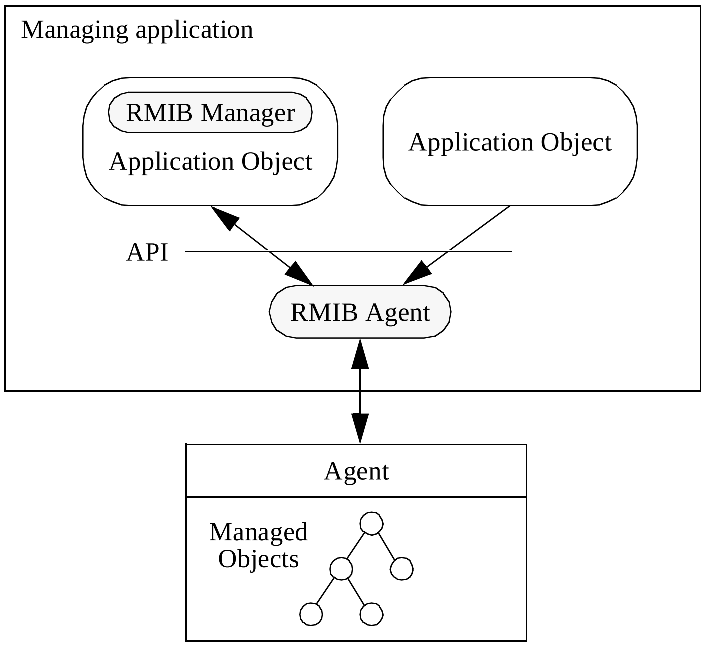

Fig. 2. Remote MIB Model Interactions

CMIS permits a number of ways to select managed objects for an operation, through a base managed object and also the type of subtree to be scoped below this. A selected subtree of managed objects can be further subjected to selection criteria, based on the logical conditions given in a filter expression. The base managed object must be identified through its distinguished name. The object identifier of the managed object class may also be provided to allow allomorphic object access. In the RMIB interface, managed object names are handled through a human readable string-based notation. For example, "*transportEntity*" and "*subsystemId=transport@entityId=isode*" refer respectively to the class and local distinguished name of the ISO transport entity object. This notation will allow the programmer to disregard completely the underlying raw CMIS API structures, leaving the necessary parsing to take place behind the interface. Using the string notation, filters can also be expressed easily and comprehensively. Finally, attribute, action and event report values use the OSIMIS OO ASN.1 API through the *Attr* (general ASN.1 type) and *AVA* (Attribute Value Assertion - type/value pair) classes.

The result returned from a request may consist of a number of managed objects. The generated managed object results may also contain attribute value assertions, as it is in the case with the M-Get, M-Set, and M-Create request primitives. Several approaches can be taken in returning the results from the interface level to the application's calling environment; having decoded the linked replies, results can be passed up either one by one or collectively in a single unit, a container. Both approaches are supported by the RMIB, transferring each response PDU into an instance of class *CMISObject*. Obviously, this class is extensive enough to capture the union of managed object parameters that can be returned in all response primitives. The individual CMISObject results may be packaged into a single containing unit, implemented by the *CMISObjectList* class.

In any design and implementation of distributed systems, it is important to allow maximum flexibility in terms of the interface interaction; that is, the interface should operate in both synchronous and asynchronous fashion. In the former, the RMIBAgent high-level operations are RPC-like in which the call will be blocked until the result or error is received. The time-out interval can be made adjustable to prevent the call from becoming blocked beyond an acceptable duration. Moreover, the total flexibility for realising complex TMN Operations Systems can only be achieved with an asynchronous interface. Using the idea of "call-back" functions, the API includes an abstract class called *RMIBManager*. This class provides a set of call-backs which must be specialised in an asynchronous manager class, and should be re-defined with the desired behaviour to respond according to different types of CMIS results. The RMIBAgent which buffers the results, after decoding and assembling the linked replies, is "informed" to be able to call back the RMIBManager.

Event reporting [X734] is fundamental to the event-driven nature of OSI management and is achieved through special support Event Forwarding Discriminator (EFD) objects. An application needs to manipulate those explicitly to request, terminate, suspend or resume reporting. In the case of the RMIB API, explicit manipulation is hidden behind the RMIBAgent class, allowing the assertion on event type, time and emitting class and instance or through an arbitrary filter, all in a string form through high-level methods that hide the related CMIS operations necessary to manipulate the EFD. An advantage of that approach is that the proliferation of EFDs in the remote MIT is avoided through centralised control, which reduces both the size of the remote agent and enables faster evaluation of event filtering criteria. Replies to confirmed event reports to acknowledge reception are also handled transparently by the RMIBAgent. As event reports are inherently asynchronous, event reporting is achieved using the RMIBManager class and the callback mechanism discussed above.

5. The Shadow MIB Access API

5.1 Rationale

One of the main considerations in the design of a manager is the dynamics of the information flow. Moreover, it is often important to minimise unnecessary manager/agent communication particularly if the communications link between the manager and agent has low bandwidth or if the amount of information requested by a manager is large. Also, since a manager normally retrieves information from an agent more frequently than it modifies it, the method of retrieval is important. A key issue here is whether the communications mechanism between manager and agent is polled or event driven. A number of factors affect this decision but two are particularly important. These are, the rate of change of information at the agent and the frequency of access by the manager. If the information at the agent changes more frequently than the rate at which the manager accesses it, a polling mechanism may be satisfactory particularly if the manager is only interested in the latest information. In this situation, unnecessary communication is minimised since the manager only accesses the agent when it needs to. Between accesses the information at the agent may go through several state changes with which the manager is not concerned.

Alternatively, if the information at the agent changes less frequently than the rate at which the manager accesses it an event driven mechanism is more suitable whereby communication only occurs when the agent notifies the manager of information changes. This can be achieved by using the ISO Object Management Function [X730] which specifies notifications for when MOs are instantiated or deleted, and when MO attributes change. In this case the manager needs to maintain a cache of information which is updated by the events from the agent. The situation is straightforward when the manager modifies MO attributes. The manager has to access the agent since both the cache and the agent have to be updated.

As explained earlier, the RMIB API provides a high level interface for interacting with the information held in an agent’s MIB. The event interface of the RMIB allows the manager to register for Object Management events but leaves the application to maintain an MIB cache. This is additional work for the application writer. What is therefore desirable is a platform function which maintains a cache of MIB information and relieves the function user from the details of maintaining them. Such a facility is the foundation of a Shadow MIB (SMIB). There are a number of options for the structure of the SMIB itself but the obvious one is to emulate the MO, attribute and MO containment tree structure of the agent’s MIB. The SMIB would then consist of Shadow MOs (SMOs).

The SMIB has a number of advantages. The primary one is that it is possible to provide an API which is easier for the application writer to use. Also, in situations where an event driven approach is unsuitable, sophisticated polling mechanisms can be provided. These are particularly useful if the manager has knowledge of the behaviour of the agent MIB. Also, if an application is managing a number of agents, there is the possibility of merging the MO containment trees of the agents into one “shadow” tree at the manager. The user of the SMIB API can therefore access the SMOs regardless of the location of the agent in which the MOs are actually located.

Work has been done in this area by the RACE NEMESYS project which produced a non-standards based shadow MIB facility called the Management Unit Information Base (MUIB) [MUIB]. Work on an OSI based SMIB facility is currently underway in the RACE ICM project.

5.2 Design Considerations

There are a number of design approaches which can be used for an SMIB facility. One of the main considerations is how much compiled knowledge is required by the application. Some options are:

> *No compiled knowledge*. This approach does not require a managing application to use GDMO information about the agent MIB. It has the advantage of being able to shadow the MIB of any agent to which it connects as long as the agent MIB uses only generic attribute types. The last point is also the main disadvantage. This approach is useful for applications such as MIB browsers.
>
> *Static knowledge*. This makes use of the GDMO already defined for the agent by compiling it to produce skeleton code for the SMIB. This allows several options in the choice of API. One possibility is to use the GDMO to produce a different C++ class for each MO with specifically named attribute access methods. This makes for a more intuitive C++ API. Since most management applications are written with semantic knowledge of the agent's functionality, it is probably not a disadvantage for the application writer to have to use the agent's GDMO.

As previously mentioned, work is underway in the RACE ICM project. The approach being used is that of no compiled knowledge. Figure 3 gives an overview of the design being adopted.As can be seen, the design builds upon the RMIB facility and is centred around two C++ classes - an SMIB Agent and Shadow Managed Object (SMO). The API to the SMIB facility is provided by both these classes. In general, operations which apply to particular SMOs or a subtree of SMOs are performed through the SMO API. Other operations are performed through the SMIBAgent API.

At initialisation time the application instantiates a single SMIB Agent and invokes it in order to instantiate the containment tree of SMOs which comprise the SMIB. Event Forwarding Discriminators [X734] and Logs [X735] are not shadowed. Once instantiated, the SMIB can be navigated using C++ pointers to objects. Scoping and filtering of sub-trees can be performed from any SMO in the tree.If the agent MIB supports object management events the application can use the SMIB Agent to initiate automatic updates of the SMIB. This means the SMIB will be updated whenever MOs are created or deleted, and MO attributes change. The SMO API also supports the option of synchronously updating the SMO whenever any of its attributes are read.

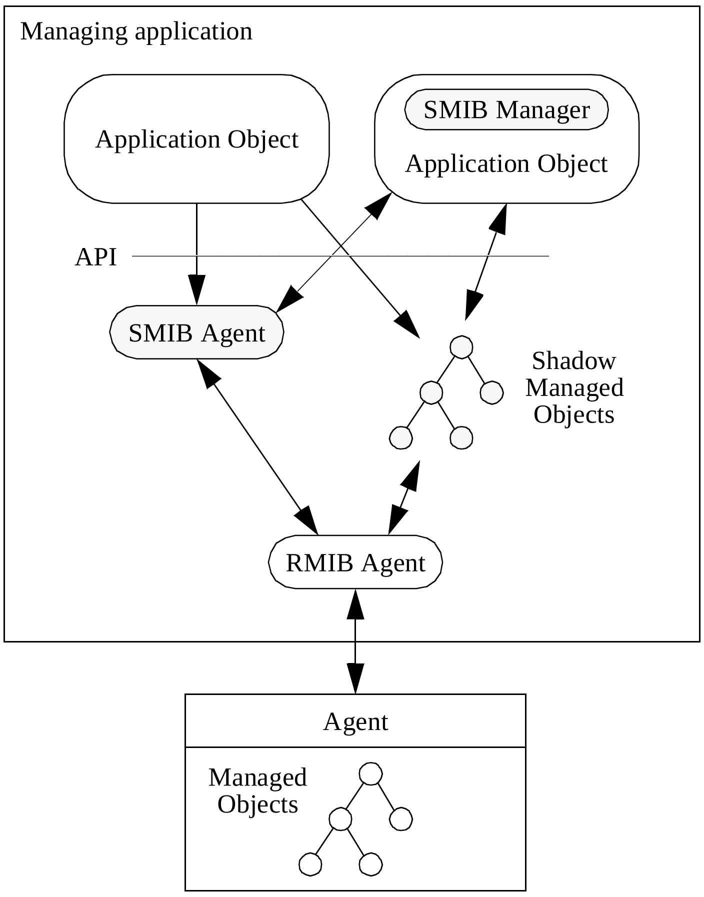

Fig. 3. Shadow MIB Model Interactions

Both synchronous and asynchronous interfaces are allowed when accessing MO attributes. Asynchronous invocations use the asynchronous invocation facility of the RMIB where the application object is called back when the response arrives. The main difference is that the callback functions are provided through the application object inheriting from a SMIB Manager class instead of a RMIB Manager class.

Provision of an SMIB API also allows a number of polling options to be offered. These include polling at regular intervals, polling according to some predominate schedule, and adaptive polling. Adaptive polling could vary it’s frequency based on, for example, the number of times an attribute has changed value in the previous ten polls, or the rate of change of a numeric value based on previous polls.

6. Discussion and Conclusions

[References

[X701] ITU X.701, Information Technology - Open Systems Interconnection - Systems Management Overview, 7/91

[M3010] ITU M.3010, Principles for a Telecommunications Management Network, Working Party IV, Report 28, 12/91

[X710] ITU X.710, Information Technology - Open Systems Interconnection - Common Management Information Service Definition, Version 2, 7/91

[XOpen] X/Open, OSI-Abstract-Data Manipulation and Management Protocols Specification, 1/92

[Pav93a] Pavlou, G., S. Bhatti and G. Knight, OSIMIS User Manual Version 1.0 for System Version 3.0, 02/93

[Pav93b] Pavlou G., The OSIMIS TMN Platform: Support for Multiple Technology Integrated Management Systems, Proceedings of the 1st RACE IS&N Conference, Paris, 11/93

[Strou] Bjarne Stroustrup, The C++ Programming Language, Addison-Wesley, Reading, MA, 1986

[X500] ITU X.500, Information Processing, Open Systems Interconnection - The Directory: Overview of Concepts, Models and Service, 1988

[X720] ITU X.720, Information Technology - Structure of Management Information - Part 1: Management Information Model, 8/91

[MUIB] Chapter 10, Experiment 3 Design, NEMESYS RACE Project 1005, ref. 05/DOW/SAR/DS/B/024/a1, 5/92

[X730] CCITT Recommendation X.730 (ISO 10164-1) Information Technology - Open Systems Interconnection - Systems Management - Part 1: Object Management Function (for CCITT Applications), 10/91

[X734] CCITT Recommendation X.734 (ISO 10164-5) Information Technology - Open Systems Interconnection - Systems Management - Part 5: Event Report Management Function, 8/91

[X735] CCITT Recommendation X.735 (ISO 10164-6) Information Technology - Open Systems Interconnection - Systems Management - Part 6: Log Control Function, 6/91